โปรล็อก ปาสคาล โปรล็อก

การค้นในทางกว้าง

บุญเลิศ เอี่ยมทัศนา

ในไมโครฯฉบับที่ 45 ได้กล่าวถึงการค้นหาคำตอบว่าทำได้หลายแบบ แบบที่ใช้กันมากที่สุดคือการค้นหาในทางลึกตามที่ได้กล่าวแล้ว กับการค้นหาในทางกว้างซึ่งจะนำมากล่าวในฉบับนี้ แต่ก่อนที่จะกล่าวต่อไปจะขอแยกโปรแกรมเก่าที่ใช้งานร่วมกันได้ มาเป็นไฟล์อิสระและเพิ่มโปรแกรมย่อยเพื่ออำนวยความสะดวกก่อน

เตรียมไฟล์ใหม่

มีโปรแกรมย่อยบางตัวในโปรแกรม DepthFirstSearch ที่สามารถนำมาใช้ในการค้นหาในทางกว้างได้ และจะใช้ต่อไปในงานอื่นด้วย ซึ่งควรจะแยกไว้ในไฟล์ต่างหากแล้วนำมารวมกับไฟล์ใหม่ด้วยการผนวก (include) ในขั้นตอนของการคอมพายล์ ขอให้ดำเนินการดังนี้

- ก็อบปี้โปรแกรม DepthFirstSearch จากไฟล์เก่าลงไฟล์ใหม่ให้ชื่อว่า SRCH1.ICL (ด้วย COPY หรือเมื่อรันเทอร์โบแล้วใช้ Ctrl-K , R) ซึ่งต่อไปนี้จะถือว่ารันเทอร์โบและกำลังอีดิตไฟล์ SRCH1.ICL อยู่

- ลบบรรทัดที่มีคำว่า PROGRAM และ USES

- ตัดฟิลด์ pass ออกจากแบบข้อมูล RoadT

- ตัดโพรซีเดอร์ต่อไปนี้ออกคือ DoAssert, SearchRoad, ClearPass, DisplayResult และ DoQuestion (คงเหลือ GetSentence, GetWord, AssertData, FindMid และ MoreRoad)

- แก้ FindMid ให้เป็นไปตามรูปที่ 1 และใน MoreRoad ให้ตัด AND และนิพจน์หลัง AND ออก

- ป้อนโพรซีเดอร์ตามรูปที่ 2 ต่อท้าย

- คอมพายล์ทดสอบโดยป้อนคำว่า begin end. ต่อท้าย เมื่อคอมพายล์ผ่านแล้วให้ตัด begin end. ออก เก็บลงดิสก์

ผลของการปรับปรุง

โปรแกรมตามรูปที่ 2 ที่ให้ป้อนต่อท้ายจะช่วยในการดำเนินการดังนี้

- DoSave จะเก็บข้อมูลในฐานข้อมูลลงไฟล์ ROAD.TXT

- DoLoad จะโหลดข้อมูลจากไฟล์ ROAD.TXT ลงฐานข้อมูล โดยยกเลิกข้อมูลในฐานข้อมูลเดิมทั้งหมด

- DoList แสดงข้อมูลในฐานข้อมูล

- DoCommand ควบคุมการทำงานของโพรซีเดอร์ทั้งสามนี้ รวมทั้งยุติการรัน

จากการปรับปรุงโปรแกรมดังกล่าวทำให้เราสามารถดำเนินการกับฐานข้อมูลได้ 3 ลักษณะคือ

- เขียนไว้ในโปรแกรมใหม่ดังที่เคยใช้ DoAssert

- ป้อนข้อมูลตรงจากคีย์บอร์ด เช่น a b 100.

- สร้างไฟล์ ROAD.TXT ไว้ก่อนด้วยเทอร์โบหรือไซด์คิก แล้วใช้คำสั่ง load โหลดลงฐานข้อมูล ไฟล์นี้เราสามารถเปลี่ยนแปลงแก้ไขต่อเติมได้ตลอดเวลา

จะเห็นว่าเป็นการอำนวยความสะดวกให้อย่างมากด้วยโปรแกรมสั้น ๆ

รูปที่ 1

```
FUNCTION  FindMid(src : NameT; var mid : NameT;
                                var idx : Word) : Boolean;
VAR   i : Word;
BEGIN
  FindMid := TRUE;
  FOR  i := idx TO RP  DO
    IF  roads[i].frm = src  THEN
      BEGIN  mid := roads[i].too;  idx := i;  Exit   END;
  FindMid := FALSE;
END;
```

และจากโปรแกรมที่ให้ต่อเติมนี้ทำให้มีคำสั่งใช้ 4 คำสั่งคือ save, load, list และ quit ให้ใช้คำสั่งนี้โดยตรงปิดท้ายด้วยเครื่องหมาย ! เช่น ต้องการยุติการรันให้ป้อนว่า quit!

สำหรับโปรแกรมหลักที่จะใช้ต่อไปเป็นดังรูปที่ 3 ซึ่งจะต้องนำไปป้อนต่อท้ายโปรแกรมทุกโปรแกรมที่จะกล่าวถึงต่อไป

การค้นในทางกว้าง (Breadth first search)

การค้นในทางกว้างจะทำการค้นหาข้อมูลลึกลงไปทีละระดับ เช่น ตามรูปที่ 4 ในการค้นครั้งแรกจะได้เมือง b, m และ x มาตรวจสอบว่าเป็นเมืองปลายทางหรือไม่ ถ้าไม่ใช่จะค้นลึกลงไปอีกระดับหนึ่งได้เมือง c, y และ z ในการค้นครั้งต่อไปจะได้เมือง d และ e จึงเหมาะสำหรับฐานข้อมูลในลักษณะดาวกระจาย เช่น สายการบินระหว่าง

รูปที่ 2

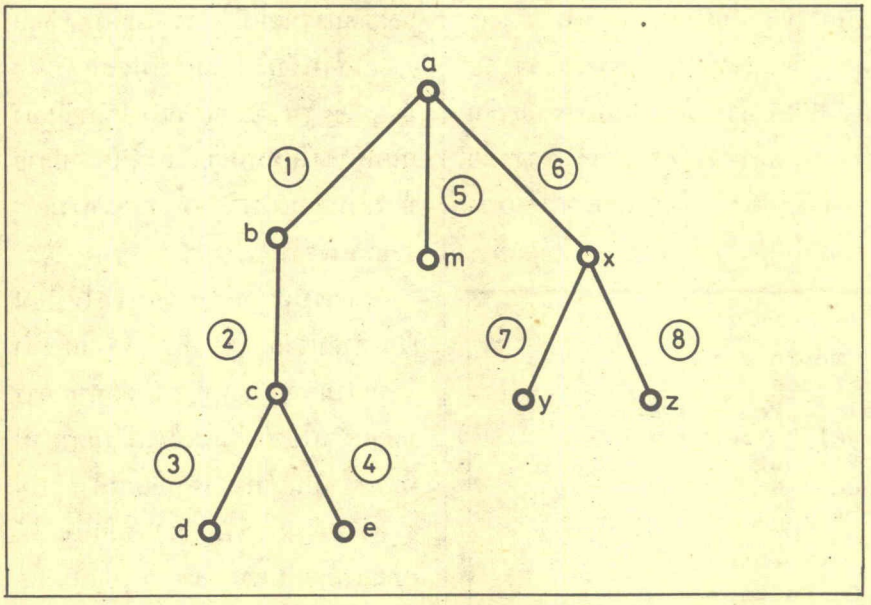
```
PROCEDURE  DoSave;
VAR   fi : Text;  idx : Word;
BEGIN
  Assign(fi,'ROAD.TXT');  ReWrite(fi);
  FOR  idx := 1 TO RP  DO
    WITH  roads[idx]  DO
      WriteLn(fi, frm,' ',too,' ',dist,'.');
  Close(fi);
END;


PROCEDURE  DoLoad;
VAR   fi : Text;  rstr : String;
BEGIN
  RP := 0;
  Assign(fi,'ROAD.TXT');
  {$I-}  ReSet(fi);  {$I+}
  IF  IOresult <> 0  THEN  WriteLn('Can''t find file ROAD.TXT')
  ELSE
    BEGIN
    WHILE  NOT EOF(fi)  DO
      BEGIN   ReadLn(fi,rstr);  AssertData(rstr)  END;
    Close(fi);
    END;
END;


PROCEDURE  DoList;
VAR    idx : Word;
BEGIN
  FOR  idx := 1 TO RP  DO
    WITH  roads[idx]  DO
      WriteLn(':', frm,' ',too,' ',dist,'.');
END;


PROCEDURE  DoCommand(astr : String);
BEGIN
  IF  astr = 'quit.'  THEN  quit := TRUE
  ELSE IF  astr = 'save.'  THEN  DoSave
  ELSE IF  astr = 'load.'  THEN  DoLOad
  ELSE IF  astr = 'list.'  THEN  DoList
END;
```

รูปที่ 3

```
BEGIN  { MAIN }
  ClrScr;  RP := 0;  quit := FALSE;
  REPEAT
    Write(':');
    GetSentence(rdestr,lstch);
    IF  lstch = '!'  THEN  DoCommand(rdestr)
    ELSE  IF  lstch = '?'  THEN  DoQuestion(rdestr)
    ELSE  IF lstch = '.'  THEN  AssertData(rdestr)
    ELSE  WriteLn('I don''t know this word');
  UNTIL  quit;
END.
```

ประเทศ มากกว่าฐานข้อมูลในลักษณะเป็นเครือข่าย เช่น เส้นทางรถเมล์ในกรุงเทพฯ

แต่ในการดำเนินการจริงนั้นจะต้องเก็บเส้นทางที่เดินผ่านไว้ด้วยทุกเส้นทาง เช่น ในการค้นถึงเมือง c, y, z จะต้องเก็บเส้นทางเป็น (a, b, c) และ (a, x, y) และ (a, x, z) เพื่อเก็บไว้แสดงผล เส้นทางแต่ละเส้นนี้จะเก็บไว้ในอะเรย์ตามแบบลิสต์ในภาษาปาสคาล แต่จะขอเรียกว่าเส้นทาง และเราจะต้องเก็บเส้นทางทั้งหมดนี้ไว้ในอะเรย์ของเส้นทางดังกล่าวอีกต่อหนึ่ง คือเป็นอะเรย์ 2 มิติ การเก็บจะเก็บในแบบคิวในภาษาปาสคาลจึงจะขอเรียกว่าคิวหรือกล่าวอีกนัยหนึ่งคือ ในคิวมีรายการเส้นทาง ในเส้นทางมีรายการเมืองที่ผ่าน



รูปที่ 4 แผนที่เส้นทางสมมติประกอบเรื่องการค้นในทางกว้าง

1. เตรียมคิวว่าง

2. หาทางแยกทีละ 1 ทางโดยที่
   - ถ้าเป็นปลายทางให้แสดงผล
   - ถ้าไม่ใช่ทางตัน ให้นำเส้นทางไปต่อท้ายคิว
   * ทำซ้ำจนครบทุกทางแยก (หากมี)

3. ถอนเส้นทางจากหัวคิวทีละ 1 เส้นทาง
   - นำมาหาทางแยกตามข้อ 2 ต่อ
   * ทำซ้ำจนคิวว่าง

รูปที่ 5

ที่เป็นเช่นนี้เพราะการค้นหาในทางกว้างจะดำเนินการในแบบ ค้นหาเมืองกลางเก็บเส้นทางไว้ในคิว(ต่อท้าย) จนครบทุกทางแยก แล้วจึงถอนเส้นทางจากหัวคิวมาค้นหาเมืองกลาง ทำต่อ ๆ กันไป เขียนเป็นขั้นตอนวิธีได้ดังรูปที่ 5

นำไปเขียนโปรแกรมได้ดังรูป A4 ในการเขียนโปรแกรมไม่ได้เก็บชื่อเมืองลงในรายการเส้นทาง เพราะจะทำให้เปลืองเนื้อที่ในหน่วยความจำ จึงเก็บเป็นดัชนีถึงเส้นทางนั้นในอะเรย์ roads แทน (ขอให้ดูการกำหนดแบบข้อมูล QueuT ในรูป A1 ด้วย)

การทำงานของโปรแกรม เริ่มด้วยการให้เส้นทางเปล่าที่ต้นคิวคือตัวแปร Q จึงไม่ได้เตรียมคิวเป็นคิวว่างอย่างที่กำหนดในขั้นตอนวิธี ทั้งนี้เพื่อใช้เป็นเงื่อนไขในการเขียนโปรแกรม ขอให้สังเกตการให้ค่า h (หมายถึงหัวคิว) และ t (หมายถึงท้ายคิว) แสดงว่ามีเส้นทางอยู่ในคิว 1 เส้นทาง

วงรอบ REPEAT คือการทำงานตามข้อ 3 ซึ่งจะต้องเป็นวงรอบนอก เริ่มจากถอนเส้นทางจากหัวคิว 1 เส้นทางให้แก่ res ซึ่งในครั้งแรกนี้จะเป็นเส้นทางว่าง แล้วให้ค่าช่องแรกของ res ค่า d (หมายถึงระดับลึก) โดยที่ได้ออกแบบไว้ให้ช่องนี้เก็บระดับลึกแทนที่จะเป็นดัชนีเหมือนช่องอื่น ๆ ในครั้งแรกนี้จึงได้ค่าเป็น 0 ทำให้ดัชนี idx ได้ค่าเป็น 0 ด้วย

ในรอบต่อไป d จะมีค่ามากกว่า 0 เช่นเป็น 1 คือเส้นทาง a-b จึงนำเมือง b มาเป็นเมืองต้นทางเพื่อค้นในระดับที่ลึกลงไปอีก สำหรับครั้งแรกนี้เมืองต้นทางคงเป็นเมืองที่ผู้ใช้กำหนดมาในคำถาม เสร็จแล้วให้ค่า idx เป็น 1 เพื่อให้ค้นจากต้นฐานข้อมูลทุกครั้งเมื่อจะเริ่มค้นหาเมืองกลางใหม่ (การใช้ idx ครั้งแรกเป็นเพียงแต่ยืมตัวแปรมาใช้)

เข้าวงรอบ WHILE โดยเรียกใช้ฟังก์ชัน FindMid แล้วนำมาเป็นเงื่อนไข ถ้าเป็นทางตันจะไม่ทำงานในวงรอบ WHILE (คือเมื่อผู้ใช้ตั้งคำถามให้เมืองต้นทางกับปลายทางเป็นเมืองเดียวกัน) และจะทำงานในวงรอบ WHILE จนหาทางแยกได้ครบทุกทางแยก เมื่อมีเมืองกลางไม่ว่าจะเป็นทางตรงหรือทางแยกจะเข้าทำงานในวงรอบ WHILE (อย่างน้อย 1 ครั้ง) นำดัชนีของเมืองนั้นเก็บใน res แล้วเพิ่มค่า idx ขึ้นหนึ่ง (เตรียมไว้หาทางแยกอื่นต่อไป)

นำเมืองกลางมาตรวจว่าเป็นเมืองปลายทางหรือไม่ ถ้าใช่ให้แสดงผล ถ้าไม่ใช่ให้ตรวจว่าเป็นทางตันหรือไม่ ถ้าไม่ใช่ทางตันให้นำเส้นทางที่บันทึกไว้ใน res ไปต่อท้ายคิว ทำจนครบทุกทางแยก เช่น เมื่อจบวงรอบ WHILE ในครั้งแรกนี้จะได้เส้นทาง a-b, a-m และ a-x แต่ a-m เป็นทางตัน ดังนั้นค่าที่เก็บในคิวจะเป็น ((1), (6)) คือเก็บเป็นค่าดัชนี

เมื่อจบจากวงรอบ WHILE จะเป็นการถอนเส้นทางจากหัวคิวมา 1 เส้นทาง คือที่เพิ่มค่า h ขึ้นหนึ่ง และต้องดูประกอบกับตอนต้นของวงรอบ REPEAT ด้วย จะดำเนินการซ้ำอยู่เช่นนี้จนหาเส้นทางได้ครบทุกเส้นทาง คือวงรอบ WHILE ไม่สามารถหาเส้นทางมาต่อท้ายคิวได้ทำให้ h = t ได้ผลการรันดังรูปที่ 6

แต่ในการใช้งานจริง ตามหลักแล้วการที่ใช้การค้นหาในทางกว้างก็เพื่อต้องการคำตอบที่สั้นที่สุด ดังนั้นเมื่อได้คำตอบแล้วไม่ควรให้ค้นหาต่อไป ทำได้โดยแก้โปรแกรมตรงที่ให้แสดงผลเป็น

BEGIN DispRes(res, d+1); Exit END

และตัดบรรทัดที่มีเครื่องหมาย {#} ออก

สำหรับโปรแกรมประกอบคือ DispRes รูป A2 เนื่องจากได้ดัชนีของถนนมาจึงให้พิมพ์ชื่อเมืองต้นทางปลายทางเป็นคู่ ๆ พร้อมทั้งระยะทางและระยะทางรวม

การป้อนโปรแกรมนี้ขอให้ป้อนเรียงลำดับจาก A1 ถึง A5 แล้วนำโปรแกรมหลักในรูปที่ 3 มาต่อท้าย การทดลองรันโปรแกรมให้ป้อนถนนตามรูปที่ 7 ซึ่งเป็นเส้นทางตามแผนที่ในรูปที่ 4 ลงไฟล์ ROAD.TXT จากนี้ขอให้ทดลองกับเส้นทางหลาย ๆ รูปแบบเพื่อเพิ่มความเข้าใจ

วิธีที่สะดวกคือรันไซด์คิกก่อน รันเทอร์โบ แล้วใช้ไซด์คิกในการป้อน

```
:a z?
[[1 ]]     level_1  route = 1
[[1 ]]     level_1  route = 5
[[1 ][6 ]]     level_1  route = 6
[[6 ][1 2 ]]     level_2  route = 1 2
[[1 2 ]]     level_2  route = 6 7
from a to x distance = 6
from x to z distance = 8
   total = 14

[[1 2 ]]     level_2  route = 6 8
[]     level_3  route = 1 2 3
[]     level_3  route = 1 2 4
```

รูปที่ 6

A1

```
PROGRAM  BreadthFirstSearch(input,output);
USES     CRT;
{$I SRCH1.ICL }
TYPE  ResT = ARRAY[0..10] OF Word;
      QueueT = ARRAY[0..20] OF ResT;
VAR   Q : QueueT;
```

A2

```
PROCEDURE  DispRes(res : ResT; p : Byte);
VAR  idx,tot : Word;
BEGIN
  tot := 0;
  FOR  p := 1 TO p  DO
    BEGIN
    idx := res[p];
    WITH  roads[idx]  DO
      BEGIN
      WriteLn('from ',frm,' to ',too,' distance = ',dist);
      tot := tot + dist;
      END;
    END;
  WriteLn('   total = ',tot);  WriteLn;
END;
```

A3

```
procedure  DispTest(res : ResT; d,h,t : Byte);
var  i,j : Byte;
begin
  write('[');
  for j := h+1  to t-1  do
    begin  write('[');
           for  i := 1 to d  do
             if  Q[j,i] <> 0  then  write(Q[j,i],' ');
           write(']')                                    end;
  write(']    ');
  write('level_',d,'  route = ');
  for  d := 1 to d  do
    Write(res[d],' ');  writeln;
end;
```

A4

```
PROCEDURE  BFS(src,dst : NameT);
VAR     mid : NameT;  idx : Word;
        res : ResT;   h,t,d : Byte;
BEGIN
  Q[0,0] := 0;  h := 0;  t := 1;
  REPEAT
    res := Q[h];  d := res[0];  idx := res[d];
    IF  d > 0  THEN  src := roads[idx].too;
    idx := 1;
    WHILE  FindMid(src,mid,idx)  DO
      BEGIN
      res[d + 1] := idx;  idx := idx + 1;
{#}   res[d+2] := 0;
      IF  mid = dst  THEN  DispRes(res,d + 1)
      ELSE  IF  MoreRoad(mid)  THEN
        BEGIN  Q[t] := res;  Q[t,0] := d + 1;  t := t + 1;  END;
{#}   DISPTEST(res,d + 1,h,t);
      END;
    h := h + 1;
  UNTIL  h = t;
END;
```

A5

```
PROCEDURE  DoQuestion(astr : String);
VAR   srce,dstn : NameT;  dist : Word; idx : Byte;
      aroad : Boolean;
BEGIN
  idx := 1;  aroad := FALSE;
  srce := GetWord(astr,idx);
  dstn := GetWord(astr,idx);
  BFS(srce,dstn);
END;
```

a	b	1.
b	c	2.
c	d	3.
c	e	4.
a	m	5.
a	x	6.
x	y	7.
x	z	8.

รูปที่ 7 ข้อมูลที่ป้อนให้แก่ไฟล์ ROAD.TXT ตามแผนที่เส้นทางในรูปที่ 4

และเปลี่ยนแปลงเส้นทาง หากเพิ่มเพียงเล็กน้อยอาจป้อนโดยตรงได้

การค้นในทางกว้างกับโปรล็อก

ดังได้กล่าวแล้วว่ากลไกภายในของโปรล็อกเป็นการค้นในทางลึก ในงานที่ต้องการให้ค้นในทางกว้างเราจะต้องดำเนินการเอง คือจะต้องเตรียมลิสต์ (ในภาษาโปรล็อก) เป็นเส้นทางเทียบได้กับตัวแปร res และลิสต์ (ในภาษาโปรล็อกอีกเช่นกัน) ไว้เก็บเส้นทางเทียบได้กับตัวแปร Q

ขอให้ดูการกำหนดรูปแบบของพรีดิเคต bfs ในรูป B1 ซึ่งจะขอไล่จากซ้ายไปขวา city, city หมายถึงเมือง ต้นทางกับปลายทาง list เทียบได้กับ

res และจะขอเรียกว่าลิสต์ R ในการอ้างอิงต่อไป llist เทียบได้กับ Q จึงจะขอเรียกว่าลิสต์ Q สำหรับ list ตัวสุดท้ายสำหรับเก็บชื่อเมืองที่ค้นหาได้ในระดับนั้นทีละเมือง ซึ่งในโปรแกรมภาษาปาสคาลใช้อินเด็กซ์ d ควบคุมการทำงานในเรื่องนี้ แต่ในภาษาโปรล็อกไม่มีอินเด็กซ์จะต้องใช้ลิสต์ดำเนินการแทนและจะขอเรียกว่าลิสต์ T

จะขอยกตัวอย่างการทำงานในรูปที่ 4 ตามกรรมวิธีการค้นในทางกว้างโดยจะให้เริ่มจากเมือง a คือให้ลิสต์ R มีค่าเป็น a ส่วน Q และ T ว่าง เมื่อเริ่มค้นหาพบเมือง b จะได้ลิสต์ R มีค่าเป็น (a, b) Q มีค่าเป็น ((a, b)) ส่วนลิสต์ T จะมีค่าเป็น (b) ในการค้นต่อไปจะพบเมือง m และ x นั่นคือเมื่อจบการค้นหาในระดับนี้จะได้ค่าในลิสต์ต่าง ๆ ดังนี้

R = (a, x)

Q = ((a, b), (a, m), (a, x))

T = (b, m, x)

ต่อไปก็จะเป็นการถอนเส้นทางจากหัวคิวคือ (a, b) มาให้แก่ลิสต์ R ทำการค้นหาต่อไป เหตุที่ต้องมีลิสต์ T เพราะว่าการค้นในทางกว้างจะไปยกเลิกกลไกภายในของโปรล็อก (ขอติดไว้ก่อนครับ) เราจึงต้องบันทึกชื่อเมืองตรงทางแยกคือในแต่ละระดับไว้ เพื่อไม่ให้ย้อนกลับไปทางแยกเดิมอีก นำไปเขียนโปรแกรมได้ดังรูป B2 ซึ่งจะขอกล่าวถึงการทำงานดังนี้

เริ่มที่พรีดิเคต trip ที่จะให้ผู้ใช้ตั้งคำถามด้วยเมืองต้นทางและปลายทาง เช่น ถามว่า trip (a, y) จะเรียกใช้พรีดิเคต bfs โดยผ่านค่า เมืองต้นทางและปลายทางไปให้ และให้ลิสต์ R มีค่าเมืองต้นทาง ลิสต์ Q และ T ว่าง

พรีดิเคต bfs ประโยคแรกเทียบได้กับวงรอบ WHILE เริ่มด้วยการหาเมืองกลางหากจะนำมาตรวจดูว่าเป็น

B1

```
/* Breadth-First Search 1 */

domains
  list = city*
  llist = list*
  city = symbol
predicates
  appz(list,llist,llist)
  do_append(list,llist,llist)
  road(city,city)
  trip(city,city)
  dsp_res(city,city,list)
  writerev(list)
  bfs(city,city,list,llist,list)
  pass(city,list)
```

B2

```
clauses

  road(a,b).
  road(b,c).
  road(c,d).
  road(c,e).
  road(a,m).
  road(a,x).
  road(x,y).
  road(x,z).


  trip(S,D) :- bfs(S,D,[S],[],[]).

  bfs(S,D,R,Q,T) :-
         road(S,M),
         not(pass(M,T)),
         dsp_res(M,D,R),
         do_append([M|R],Q,N),!,
         bfs(S,D,R,N,[M|T]).

  bfs(_,D,_,[[S1|R1]|Q1],_) :-
         bfs(S1,D,[S1|R1],Q1,[]),
         fail.

  dsp_res(D,D,R) :-
         write("the route is :- "),
         writerev(R),
         write(D),nl,!.
  dsp_res(_,_,_).

  do_append([M|R],Q,N) :-
        road(M,X),
        writerev([M|R]),
        write(" (",X,")"),nl,
        appz([M|R],Q,N).

  appz(N,[],[N]).
  appz(N,[H|T],[H|T2]) :- appz(N,T,T2).


  writerev([]).
  writerev([H|T]) :- writerev(T), write(H,' ').

  pass(X,[X|_]).
  pass(X,[_|Y]) :- pass(X,Y).
```

เมืองที่ผ่านมาก่อนหรือไม่ ถ้าไม่เคยผ่านให้นำออกแสดงด้วย dsp-res

ซึ่ง dsp-res จะตรวจว่าถึงเมืองปลายทางหรือไม่ก่อน ถ้าถึงจึงจะนำออกแสดง ขอให้สังเกตด้วยว่า dsp-res ประโยคที่สอง ทำให้พรีดิเคตนี้เป็นจริงตลอดเวลา

จากนี้จะนำไปต่อท้ายคิวด้วย do-append ซึ่งจะตรวจก่อนว่าเป็นทางตันหรือไม่ ถ้าไม่ใช่ทางตันจึงจะนำไปต่อท้ายคิวด้วย appz ที่ให้ writerev และ write ในพรีดิเคตนี้เพื่อการศึกษาเท่านั้น ผลจากการต่อท้ายคิวจะส่งกลับมาในตัวแปร N

ท้ายสุด คือการเรียกตนเองด้วยค่าเมืองต้นทางและปลายทางเดิม ได้เส้นทางในระดับนี้ในคิว Q เมื่อหมดเส้นทางในระดับหนึ่งแล้ว จะทำงานในพรีดิเคต bfs ประโยคที่ 2 เทียบได้กับวงรอบ REPEAT ขอให้สังเกตการแยกส่วนหัวออกจากลิสต์ Q เพื่อนำไปให้แก่ลิสต์ R และได้แยกส่วนหัวของลิสต์ R ซึ่งเป็นเมืองกลาง นำไปเป็นเมืองต้นทางในการค้นหาในระดับต่อไป สำหรับ Q1 คือส่วนหางของคิวที่เหลือนำไปให้ค่าแก่ Q ของการค้นครั้งต่อไป

เมื่อเรียกตนเองก็จะไปทำงานที่ bfs ประโยคแรกอีกค้นหาเมืองกลางในระดับนั้นดังกล่าวแล้ว การเรียกตนเองจะยุติเมื่อเรียกตนเองด้วยค่าในลิสต์ Q ว่าง และ bfs แรกไม่สามารถจะหาเส้นทางมาต่อท้ายคิวได้คือหมดเส้นทางเมื่อมาถึง bfs ที่ 2 ก็จะเป็นเท็จเพราะลิสต์ว่างไม่สามารถแยกเป็นส่วนหัวและส่วนหางได้ หรือเมื่อพบเมืองปลายทางแล้วโปรล็อกจะให้ยุติการค้นหา

จากนี้ก็จะเป็นการส่งกลับ มาทำงานที่ fail ซึ่งเมื่อใช้คำนี้หากเป็นการยุติการเรียกตนเองเนื่องจากโปรล็อกให้ยุติ (พบเมืองปลายทางแล้ว) จะเป็นการบังคับให้โปรล็อกค้นหาคำตอบคือเรียกตนเองต่อไป จุดประสงค์เพื่อให้แสดงเส้นทางที่จะไปยังเมืองปลายทางทุกเส้นทาง และในกรณีนี้เพื่อแสดงการทำงานของโปรแกรมนี้ด้วย ดังผลการรันในรูปที่ 8

แต่ในการใช้งานจริงไม่ควรใช้คำว่า fail ในการค้นหาในแบบทางกว้าง เพราะเมื่อเราใช้การค้นหาในแบบทางกว้างก็เพื่อต้องการคำตอบที่สั้นที่สุด หากต้องการทุกคำตอบควรใช้การค้นในทางลึก ซึ่งจะได้คำตอบเร็วกว่าและใช้สแต็กน้อยกว่า

```
Goal: trip(a,z)
a b  (c)
a x  (y)
a b c  (d)
the route is :- a x z
No
Goal:
```

รูปที่ 8

ขอใช้หนี้ที่ติดไว้

ผมได้ติดหนี้ไว้หลายแห่งและเขียนข้อความกำกวมไว้ในที่หลายแห่งเช่นกัน ทั้งหมดนี้เพราะว่าโปรล็อกโดยเฉพาะเทอร์โบโปรล็อกไม่ได้ค้นหาคำตอบในแบบทางลึกแท้ ๆ มีทางกว้างผสมด้วย

ซึ่งถ้าคุณทำการแทร็กซิงโปรแกรมค้นในทางลึก (ฉบับที่แล้ว) จะเห็นว่าโปรล็อกไม่ได้ค้นหาข้อมูลจากต้นฐานข้อมูลเสมอไป คือถ้าเป็นทางแยกจะเริ่มตรงทางแยก เช่น ตามรูปที่ 4 เมื่อถอยกลับมาเมือง c จะไป e โปรล็อกจะเริ่มต้นที่ "c e 4." (รูปที่ 7) โดยตรงไม่ได้ไปเริ่มต้นค้นที่ต้นฐานข้อมูลคือที่ "a b 1." ใหม่ ดังที่ผมเคยกล่าวมาและโปรล็อกก็ไม่ได้ใช้วิธีหมายรอยเท้าซึ่งต้องเสียเวลาทั้งในตอนหมายและตอนลบด้วย

โปรล็อกทำได้อย่างไร หลังจากที่เราทราบหลักการค้นหาคำตอบทั้งสองแบบแล้ว ขอให้พิจารณาดูใหม่ครับว่าในการค้นหาคำตอบในทางลึกนั้น เราจะพบสถานะอะไรบ้างและจะให้ดำเนินการอย่างไร จะพิจารณาได้ดังนี้

1. เป็นเมืองปลายทาง = ให้แสดงผล
2. เป็นทางตรง = ให้เดินตรงไป
3. เป็นทางแยก = ให้จำทางแยกทุกทางไว้ แล้วเดินไปทางซ้าย
4. ถ้าเป็นทางตัน = ให้ถอยกลับไปที่ทางแยก แล้วเดินไปทางขวา

* ทำซ้ำจนกระทั่งไม่มีทางเดิน

นำไปเขียนโปรแกรมได้ดังรูป C1 ถึง C4 (ซึ่งขอเรียนให้ทราบว่าในโปรแกรมนี้บรรทัดที่ขึ้นต้นด้วย {#} คือบรรทัดที่แทรกเข้ามาเพื่อแสดงการทำงานของโปรแกรมเท่านั้น) ขอให้ดูการกำหนดแบบข้อมูล BranchT ในรูป C1 ก่อน เป็นแบบข้อมูลที่จะเก็บทางแยกไปถนนสายต่าง ๆ ในแต่ละทางแยกโดยจะเก็บเป็นดัชนีของถนนนั้น ๆ (นั่นคือในแต่ละแยกจะมีทางแยกได้ไม่เกิน 10 ทาง)

การค้นหาเริ่มจาก DoQuestion ในรูป C4 ซึ่งจะเรียกใช้ DFS โดยผ่านค่าเมืองต้นทางและปลายทางไปให้ ในโพรซีเดอร์ DFS รูป C3 ขอให้สังเกตว่าได้กำหนดตัวแปร bran และ bp เพื่อเก็บทางแยกใน bran ส่วน bp เป็นดัชนีสำหรับอะเรย์นี้

การทำงานเริ่มด้วยการเก็บเมืองต้นทางลงสแต็ก stack กำหนดไว้ใน

SRCH1.ICL ซึ่งถ้าเป็นรอบการทำงานต่อไปก็จะเป็นเมืองกลาง เป็นการบันทึกเส้นทางตามที่เดินผ่านไปเสร็จแล้วให้ตรวจว่าเมืองกลางกับเมืองปลายทางเป็นเมืองเดียวกันหรือไม่ ถ้าใช่ให้แสดงผล

จากนี้ให้หาเมืองถัดไปด้วย FindNext รูป C2 โดยผ่านค่า src ไปให้ส่วน bran กับ bp ถือว่าเป็นค่าว่างเพื่อให้ FindNext ส่งกลับมา

```
PROGRAM  DepthFiestSearch3(input,output,fi);
USES  CRT;
{$I SRCH1.ICL }
TYPE   BranchT = ARRAY[1..10] OF Word;
VAR    ch : Char;


PROCEDURE  DispRes(SP : Byte);
VAR  i : Byte;
BEGIN
  Write(' the route is:-  ');
  FOR   i := 1 TO SP  DO  Write(stack[i],' ');
  WriteLn;
END;
```

C1

```
FUNCTION  FindNext(src : NameT; var bran : BranchT; var bp : Byte) : Boolean;
VAR  idx : Byte;
var  i : Byte;
BEGIN
  idx := 0;  bp := 0;
  WHILE  (idx < RP)  DO
    BEGIN
    idx := idx + 1;
    IF  src = roads[idx].frm  THEN
      BEGIN  Inc(bp);  bran[bp] := idx   END;
    END;
{#}  write(' <');  for i := 1 to bp do write(bran[i],' ');  writeln('> ');
  bran[bp + 1] := 0;   FindNext := (bp > 0);
  bp := 0;
END;
```

C2

```
PROCEDURE  DFS(src,dst : NameT);
VAR   bran : BranchT;  bp : Byte;
var  i : Byte;
BEGIN
  stack[SP] := src;  SP := SP + 1;
{#}  write(' [');  for i := 1 to SP-1 do write(stack[i],' ');  write('] ');
  IF  src = dst  THEN   DispRes(SP - 1);
  IF  FindNext(src,bran,bp)  THEN
    WHILE  bran[bp+ 1] <> 0  DO
    BEGIN  bp := bp + 1;  src := roads[bran[bp]].too;
           DFS(src,dst)                                END;

  SP := SP - 1;
END;
```

C3

ใน FindNext จะค้นหาเมืองถัดไปทุกเมืองคือทางแยกทั้งหมดที่มีด้วยวงรอบ WHILE เก็บใน bran เมื่อออกจากวงรอบ WHILE ให้ปิดท้ายรายการทางแยกด้วย 0 เพื่อใช้เป็นเงื่อนไขในการเขียนโปรแกรมต่อไป กับให้ FindNext เป็นจริงหากมีเมืองถัดไป แล้วให้ bp = 0

กลับมาที่ DFS หากมีเมืองถัดไปก็จะนำมาหาทางตรงหรือทางแยกต่อไปด้วยวงรอบ WHILE ตรงนี้เป็นการรวมข้อ 2 กับข้อ 3 เข้าด้วยกัน คือถ้าเป็นทางตรงก็จะทำงานในวงรอบนี้เพียงครั้งเดียว ถ้าเป็นทางแยกก็จะทำเป็นจำนวนครั้งเท่ากับทางแยก โดยเรียกตนเองทุกครั้ง (ทำ FindNext อีก)

หากทางใดเป็นทางตันก็จะส่งกลับถอนค่าคืนจากสแต็ก กลับไปตรงทางแยกคือในวงรอบ WHILE เพื่อจะเดินไปในทางแยกอื่นต่อไป หากแผนที่เส้นทางเป็นไปตามรูปที่ 4 และผู้ใช้ถามเส้นทาง a ไป m จะได้การดำเนินการดังนี้

```
PROCEDURE  DoQuestion(astr : String);
VAR   srce,dstn : NameT;  dist : Word; ii : Byte;

BEGIN
  ii := 1;  SP := 1;
  srce := GetWord(astr,ii);
  dstn := GetWord(astr,ii);
  DFS(srce,dstn);
  WriteLn('No (more) road');
END;
```

C4

- เมื่อ DoQuestion เรียกใช้ DFS แล้วค่าในสแต็กจะเป็น (a) เรียก FindNext เมื่อกลับคืนมาค่าใน bran จะเป็น (1, 5, 6) มาทำงานในวงรอบ WHILE เรียกตนเองด้วย src = b

- ทำให้ค่าในสแต็กเป็น (a, b) เรียก FindNext เมื่อกลับคืนมาค่าใน bran จะเป็น (2) แต่เป็น bran คนละตัวกับที่เก็บค่า (1, 5, 6) เรียกตนเองอีกด้วย src = c

-ได้ค่าในสแต็กเป็น (a, b, c) เรียก FindNext ได้ค่าใน bran ใหม่เป็น (3, 4) เรียกตนเองด้วย src = d

- ได้ค่าในสแต็กเป็น (a, b, c, d) ปรากฏว่าทางตัน ถอน d ออกจากสแต็ก แต่การทำงานยังอยู่ในวงรอบ WHILE จึงเรียกตนเองด้วย src = e (จากเส้นทางหมายเลข 4 ใน bran)

- ได้ค่าในสแต็กเป็น (a, b, c, e) ปรากฏว่าทางตัน ถอน e ออกจากสแต็ก กลับไปวงรอบ WHILE ของครั้งนี้ปรากว่าไม่มีทางแยกอื่นอีก จึงถอน c ออกจากสแต็ก แล้วถอน b จากสแต็กเพราะไม่มีทางแยกอื่นอีกเช่นกัน

- กลับไปถึง a ปรากฏว่ามีทางแยกอื่นคือค่าใน bran ของครั้งนี้เป็น (1, 5, 6) เมื่อเพิ่มหนึ่ง bp จะได้เส้นทางเป็น 5 เรียกตนเองด้วย src = m

- ได้ค่าในสแต็กเป็น (a, m) ปรากฏว่า m เป็นเมืองปลายทางจึงให้แสดงผลเป็น a ไป m โดยตรง

- เสร็จแล้วให้หาทางแยกอีก ปรากฏว่า m เป็นทางตันจึงถอนจากสแต็ก การทำงานต่อไปก็จะเดินไปทาง ax เช่นเดียวกับที่กล่าวมาแล้ว และยุติการรันเมื่อไม่มีเส้นทางจะไปต่อ

ได้แสดงผลการรันไว้ในรูปที่ 9 ชื่อเมืองในวงเล็บก้ามปูคือ ชื่อเมืองที่บันทึกไว้ใน stack ส่วนที่อยู่ในวงเล็บแหลมคือ หมายเลขเส้นทางตรงทางแยกซึ่งบันทึกไว้ใน branch ซึ่งให้แสดงเพื่อการศึกษาเท่านั้นสำหรับผลการรันจริง ๆ คือที่อยู่หลังประโยคว่า the route is:-

ผมกล่าวมายืดยาวทั้งที่มองเหมือนเรื่องธรรมดา แต่ผมก็ใช้เวลากว่า 2 ปี เพื่อจะหาคำตอบว่า โปรล็อกกระทำซ้ำหรือย้อนรอย (back tracking) ได้อย่างไร และก็ไม่มีกล่าวในหนังสือภาษาอังกฤษ ไม่ว่าหนังสือที่เกี่ยวกับภาษาคอมพิวเตอร์ หรือขั้นตอนวิธีและโครงสร้างข้อมูลเท่าที่ผมจะค้นหาได้เลย อย่างดีก็ใช้วิธีหมายรอยเท้าอย่างที่ผมนำมาแสดงในตอนที่แล้ว

ผมได้เรื่องนี้มาด้วยความบังเอิญ ขณะที่ผมเขียนโปรแกรมภาษาปาสคาลเพื่อแสดงการค้นหาในทางกว้างก็นึกออก (มันคงง่ายเช่นนี้จึงไม่มีใครกล่าวถึงกัน แต่สำหรับผมแล้วถือว่าเป็นงานชิ้นเอกในรอบปีทีเดียว) ขอให้รันโปรแกรมนี้เทียบกับการแทร็กโปรแกรมภาษาโปรล็อก (รูปที่ 9 ในตอนที่แล้ว) จะเห็นว่ากรรมวิธีในการย้อนรอยเหมือนกัน แต่ทำไมผมถึงกล่าวว่าไม่ทราบว่าโปรล็อกทำได้อย่างไร ก็ตอนนั้นผมไม่ทราบจริง ๆ ครับ

```
:a z?
 [a ]  <1 5 6 >
 [a b ]  <2 >
 [a b c ]  <3 4 >
 [a b c d ]  <>
 [a b c e ]  <>
 [a m ]  <>
 [a x ]  <7 8 >
 [a x y ]  <>
 [a x z ]  the route is:-  a x z
 <>
No (more) road
:
```

รูปที่ 9

สรุปเรื่องในภาษาปาสคาล

เท่าที่ผมกล่าวมาใน 2 ฉบับนี้ สำหรับทางด้านภาษาปาสคาลคือ ได้นำขั้นตอนวิธีของการค้นหาคำตอบในแบบทางลึกและทางกว้างมากล่าวถึงซึ่งพอสรุปได้ว่า

การค้นในทางลึก

- ให้เตรียมตัวแปรในแบบลิสต์สำหรับเก็บทางแยกทุกแยก ในแต่ละเส้นทาง ด้วยการเรียกตนเองเพื่อป้องกันการย้อนกลับไปในทางเดิม

- ให้เตรียมตัวแปรในแบบสแต็กสำหรับบันทึกชื่อเมืองที่ผ่านไปตามลำดับ ซึ่งจะถูกถอนออกเมื่อเป็นทางตัน อันเป็นผลมาจากการเรียกตนเอง หากถึงเมืองปลายทางค่าในตัวแปรนี้คือคำตอบ

การค้นในทางกว้าง

- ให้เตรียมตัวแปรในลักษณะคิวเพื่อเก็บเส้นทางไปยังเมืองกลางลึกลงไปทีละระดับ เมื่อเสร็จสิ้นการค้นหาในระดับหนึ่ง ๆ แล้วยังไม่พบเมืองปลายทาง ให้ถอนเส้นทางจากหัวคิวมาตรวจหาเส้นทางลึกลงไปอีกระดับหนึ่ง ซึ่งถ้าพบเมืองปลายทาง เส้นทางที่พึ่งถอนจากคิวคือคำตอบ หากไม่ใช่เมืองปลายทางและไม่ใช่ทางตันให้นำไปต่อท้ายคิว ถ้าเป็นทางตันไม่ต้องดำเนินการใดเท่ากับถอนเส้นทางนั้นออกจากคิว

เมื่อนำไปใช้ในงานอื่นก็ขอให้นึกเทียบถึงเมืองต้นทางกับปลายทางคือจาก-ถึง สำหรับเรื่องของโปรล็อกขอพักไว้ก่อนครับ